PARALLELIZED RANDOM WALK ALGORITHM FOR BACKGROUND SUBSTITUTION ON A MULTI-CORE EMBEDDED PLATFORM

Yutzu Lee[1], Chen-Kuo Chiang[2], Yu-Wei Sun[1], Te-Feng Su[2], Shang-Hong Lai[1,2]

Institute of Information Systems and Applications[1], Department of Computer Science[2]
National Tsing-Hua University, Hsin-Chu, Taiwan
s9965508@m99.nthu.edu.tw, {ckchiang, tfsu, lai}@cs.nthu.edu.tw, tokillhe@yahoo.com.tw

ABSTRACT

Random Walk (RW) is a popular algorithm and can be applied to many applications in computer vision. In this paper, a fast algorithm is proposed to solve the large linear system in RW based on adapting the Gauss-Seidel method on a multi-core embedded system. Two tables, *TYPE* and *INDEX*, are introduced to fast locate the required data for the close-form solution. The computational overhead, along with the memory requirement, to solve the linear system can be reduced greatly, thus making the RW algorithm feasible to many applications on an embedded system. In addition, the proposed fast method is parallelized for a heterogeneous multi-core embedded platform to make the most use of the benefits of the system architecture. Experimental results show that the computational overhead can be significantly reduced by the proposed algorithm.

Index Terms— Random walk, image segmentation, background substitution, parallelization, multi-core embedded system.

1. INTRODUCTION

Random Walk (RW) algorithm, firstly proposed by Wechsler and Kidode [1] for texture discrimination, has been applied to various problems recently, such as image segmentation [2], colorization [3] and stereo matching [4]. As humans are the principal subject of images, segmentation of humans is important for certain tasks of image editing, such as object selection and image synthesis. Due to the strong demand and rapid development of image editing and processing, there has seen increasing attention to accurate foreground extraction from images. In [5], RW is adopted to extract foreground objects by background subtraction using approximate seed points.

The recent emergence of multi-core-embedded systems enables more and more image/video applications. Multi-core architecture provides cost-effective and energy-efficient computations. It is suitable for heavy-workload applications, such as the aforementioned image segmentation problems.

In this paper, we propose a fast method for applying the Random Walk algorithm for background substitution on a heterogeneous multi-core embedded system. By building the proposed *Type* and *Index* tables, the computational overhead, along with the memory requirement, to solve a large linear system in RW can be reduced greatly, thus making the RW algorithm feasible to many applications on an embedded system. In addition, a parallel algorithm has been developed for the proposed fast RW to utilize the benefits of the architectural features of a multi-core system. We demonstrate our method by the background substitution application in a multi-core embedded platform, called PAC Duo (PAC means Parallel Architecture Core) [6]. This multi-core platform contains an ARM9 processor and two PACDSP cores. The PACDSP core was developed with innovative distributed ping-pong register file and variable-length VLIW encoding techniques.

2. RANDOM WALK ALGORITHM REVISIT

A graph $G = (V, E)$ has vertices $V = \{v_i\}_{i=1,\dots,N}$ and edges $e \in E \subseteq V \times V$. A weighted graph has a value assigned to each edge, and it is called a weight. The weight between two vertices v_i and v_j is denoted by w_{ij}. The degree of a vertex is $d_i = \sum w_{ij}$. In this work, the graph is assumed to be undirected. In other words, $w_{ij} = w_{ji}$.

The energy function to be minimized in RW [7] is:

$$E_{RandomWalks} = \sum_{e_{ij} \in E} w_{ij}(x_i - x_j)^2 = \boldsymbol{x}^T \boldsymbol{L}\boldsymbol{x}, \quad (1)$$

where the (i,j)-th entry in **L,** denoted by L_{ij} and associated with vertices v_i and v_j, represents the combinatorial Laplacian matrix [8] which is defined by

$$L_{ij} = \begin{cases} d_i & if\ i = j, \\ -w_{ij} & if\ v_i\ and\ v_j\ are\ adjacent\ nodes, \\ 0 & otherwise, \end{cases} \quad (2)$$

Partitioning the vertices into marked node set V_M (seeds) and unmarked node set V_U, Eq.1 can be decomposed into:

$$E_{RandomWalks} = [\mathbf{x}_{\mathrm{M}}^{\mathrm{T}} \quad \mathbf{x}_{\mathrm{U}}^{\mathrm{T}}] \begin{bmatrix} \mathbf{L}_M & \mathbf{B} \\ \mathbf{B}^T & \mathbf{L}_U \end{bmatrix} \begin{bmatrix} \mathbf{x}_{\mathrm{M}} \\ \mathbf{x}_{\mathrm{U}} \end{bmatrix}, \quad (3)$$

Differentiating the above matrix form for the energy function with respect to x_U and setting it to zero yields

$$\boldsymbol{L_U x_U = -B^T x_M} \quad (4)$$

After computing the likelihood $\boldsymbol{x}_U$, each node v_i in V_U can be assigned a class label, foreground or background.

3. PROPOSED FAST RANDOM WALK ALGORITHM

Based on the above formulation, the dimension of the matrix **L** in a 256 by 256 image is 65525 by 65525. Directly solve the linear system is not feasible for an embedded system. Since the matrix **L** records only the relations between pixels and their neighbors, it is pretty sparse. A naïve method is to build **L** with link list. Here, a fast method is proposed to solve the unknowns with two additional tables. The proposed method is designed to be highly parallelizable and suitable for running on a multicore architecture. Experimental results show that the proposed method offers over an order of magnitude speed increase compared to the naïve implementation.

3.1. Gauss-Seidel Method

The Gauss-Seidel method [9] is an iterative technique for solving a system of *n* linear equations **Ax**=**b** in an iterative fashion, which uses previously computed results to update the solution as soon as they are available. The update equation can be simply written as:

$$x_i^{(k)} = \frac{b_i - \sum_{j<i} a_{ij} x_j^{(k)} - \sum_{j>i} a_{ij} x_j^{(k-1)}}{a_{ii}} \quad (5)$$

where a_{ij} and b_i are the entry from matrix **A** and vector **b**, respectively, and *k* is the iteration number.

3.2. Data Arrangement

Here, we use a simple 3x2 image as an example to show how the sparse linear system of random walk can be solved efficiently. In the 3x2 image, pixels marked by blue circle shows the foreground seed points, and the background seed is marked by green circle. If the neighboring relation is defined by 1:upper, 2:left, 3:right and 4:down neighbor, the weight can be built by a 6x4 matrix, as shown in Fig. 1. Empty entries mean the neighbors do not exist. Degree, the pixel values of seeds and unseeds are stored as a 1-D array.

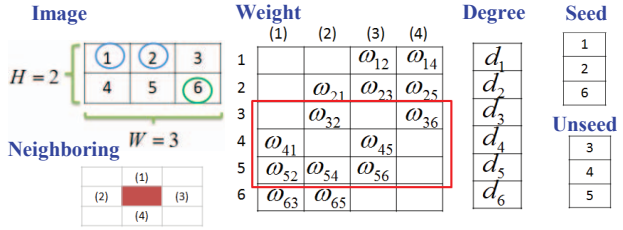

Fig. 1. A simple example shows the image and the corresponding data structure.

According to Eq. 4, the linear system has three unknowns: x_3, x_4 and x_5. Based on Eq. 5 and the existence of the four neighbors, the solution is shown in Fig. 2. Note that, to obtain the solution, it requires the data from seed (X_1, X_2, X_6) and unseed (x_3, x_4, x_5). Conceptually, the required data is listed in the *Solution Table*. In practical, it needs to scan both *Seed* and *Unseed* array to retrieve the required data index and the pixel values, thus very time-consuming. In the next section, two tables are proposed to realize the *Solution Table* efficiently.

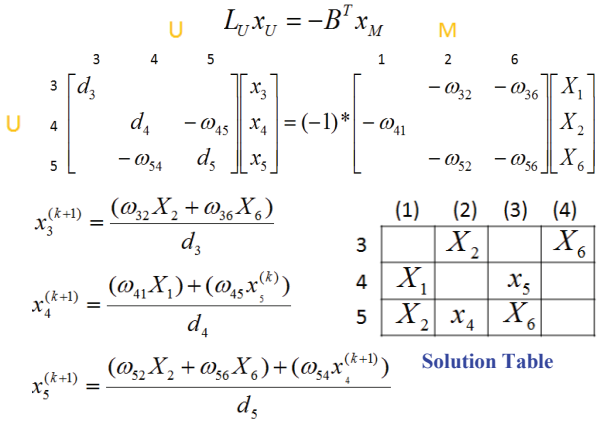

Fig. 2. A simple example of the linear system and the Solution table for the Random Walk algorithm.

3.3. TYPE and INDEX Tables

To fast retrieve the data required for the closed-form solution, we first define the data type: 1 for unseed points, 2 for seed points and 0 for non-existence neighbors. Compared to the *Solution Table*, a *TYPE Table* can be created as shown in Fig.3. Another table, *INDEX Table*, records the order of seeds and unseeded points in their own 1-D arrays. For example, seeded point x_2 has the order of 2 in the *Seed* array and the order of unseeded point x_5 is 3 in the *Unseed* array.

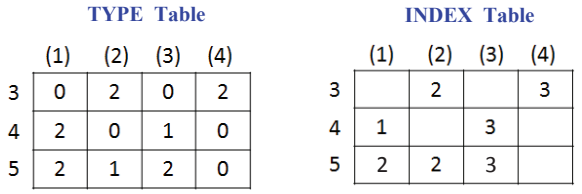

Fig. 3. A simple example of TYPE table and INDEX table.

After these two tables are established, the solution can be obtained by scanning the each row in *TYPE Table* in order. The non-zero entries indicate which data array to retrieve data (*Seed* or *Unseed*). The co-located entries in *INDEX Table* shows where to get data inside the data array by using the array index. Now, let's go back to the closed-form solution of x_3:

ALGORITHM 1: *Fast Random Walk Algorithm*

Input: Image *I*.
Initialization: Set $k \leftarrow 1$ and ε .
Calculate *Weight* and *Degree* matrices. Store seeded and unseeded points into *Seed* and *Unseed* array.
repeat {Main loop}:
  If $k == 1$, build *TYPE* and *INDEX* Tables.
    Solve $x_i^{(k)}$ by Gauss-Seidel Method.
    $k \leftarrow k + 1$. Continue.
  End if.
  Update $x_i^{(k)}$ by the solution from previous iteration.
  $k \leftarrow k + 1$.
until $\left| x_i^{(k)} - x_i^{(k-1)} \right| < \epsilon$
Assigning label to each node by the probability.

$$x_3^{(k+1)} = \frac{\omega_{32} X_2 + \omega_{36} X_6}{d_3} \qquad (6)$$

To solve x_3, we first scan the first row in *TYPE Table*. The two non-zero entries show the data should be retrieved from *Seed* array. Then, the co-located entries in *INDEX Table* provide the array index 2 and 3. Thus, X_2 and X_6 can be obtained easily and fast. We still need w_{32} and w_{36}. By storing the seeded and unseeded data from *Weight* into two separate matrices in advance (saving the data in the red frame into a separate matrix in Fig.1), w_{32} and w_{36} can be obtained easily by using the same data index of non-zero entries in *TYPE Table*. The solution is considered as the probability whether it is foreground or background points. The details are given in Algorithm 1.

3.4. Parallelization Strategy

Recall that there are two important characteristics of the Gauss-Seidel method. Firstly, the computations appear potentially to be serial. Since each component of the new iterate depends upon all previously computed components, the updates cannot be done simultaneously. Secondly, the new iterate $x^{(k)}$ depends upon the order in which the equations are examined. If this ordering is changed, the components of new iterates will also change. In other words, the solution depends on the iteration order and the equation order within each iteration. To achieve parallelization, each component is computed by using all the results from the previous iteration to reduce the data dependency.

4. PARALLELIZATION ON A MULTI-CORE EMBEDDED PLATFORM

PAC Duo evaluation board equips a multi-core SoC consisting of three processors, one ARM9 core and two DSPs. It is suitable for highly parallelizable multimedia applications in embedded systems. Fig. 4 illustrates the PACDSP architecture. All cores are connected with a AHB bus along with 256MB share memory. The DSP used in

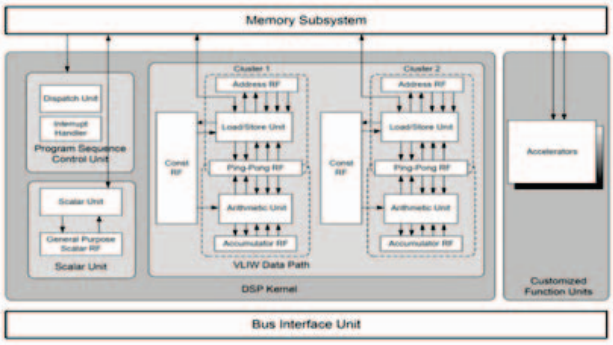

Fig. 4. PACDSP Architecture.

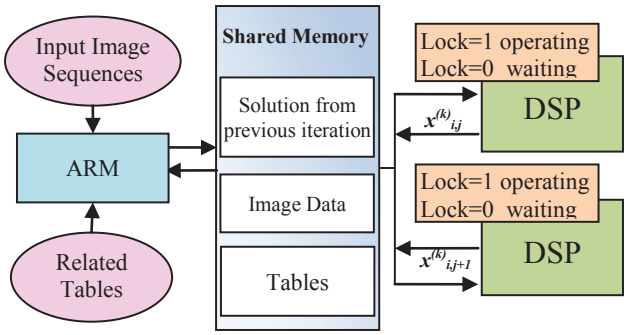

Fig. 5. The parallelization of the proposed method.

PACDuo SoC is PACDSP which is a 32-bit, fixed-point, and five-way issue VLIW DSP. PACDSP is comprised of two Load/Store Units (LSU), two Arithmetic Logic Units (ALU), and one Scalar Unit. LSU and ALU are organized into two clusters, each containing its own private register file and being capable of accessing public register file.

In this parallelization mechanism, the seed points are decided first by the initialization method. The calculation of *Weight* and *Degree* matrices are assigned to ARM processor because it requires more floating point operations. Then, PACDSP cores are dedicated to solve the linear system by the Gaussian-Seidel method. A lock mechanism is used to control synchronization between ARM processor and PACDSP cores. If the lock value is one, it indicates PACDSP core is performing the task. Otherwise, it waits until the PACDSP core is unlocked. Since each core performs the computational tasks independently, it is easy to extend this parallelization strategy to more PACDSP cores.

5. EXPERIMENTAL RESULTS

We evaluate the proposed fast RW algorithm by a background substitution application. Face detection is performed along with a human shape prior model to decide the rough area of human and the background. Then, these areas are assigned to foreground and background seed points. After applying the RW algorithm, the foreground objects can be extracted and composited with a new background.

We first compare the proposed fast RW algorithm to the baseline method and the link list method in C code

TABLE. 1. The average execution time (ms) of solving the linear system in RW algorithm by the proposed method, Baseline and Link List in ten 320x240 images in C code.

Time (ms)	Baseline	Link List	Proposed
Itr k = 0	134.89	5557.64	135.21
Itr k = 1.. N	134.89	0.5813	0.4628
Total	146700	5839.51	580.30

Fig. 6. Sample Results of the background substitution.

implementation. The baseline method solves the linear system in RW directly (conceptually retrieve data in the *Solution Table*). The second method implements the matrix **L** using link list structure. All tests are performed on an Intel Core2 CPU 6320 at 1.86 GHz without parallelization. We have tested ten 320x240 images and calculate the average time. Table 1 shows the execution time of solving the linear system in RW algorithm with the three methods. Note that since the baseline method retrieve data in the same manner, the execution time remains the same in each iteration. In Link List and the proposed method, they need to build the related structure and tables in the first iteration, thus the execution time is much slower than the other iterations. Overall, the proposed method is about 10 times faster than Link List and 252.8 times faster than the Baseline.

We also perform the experiments on the PAC Duo platform to evaluate the speedup ratio and the overall performance after parallelization. The clock frequency is set to 150 MHz. Since PACDSP and ARM9 in the PAC Duo platform support different instruction sets, the proposed system needs two sets of compilation toolkits to conduct the final executable. For the code running on ARM9, we used Sourcery G++ Lite Edition ARM 2008q3 cross-compilation toolkit to build the code. On the other hand, we used pacc32, an open64-based cross-compilation toolkit, to build the part of code that is partitioned to PACDSPs. We set the optimization level of both toolkits to O3.

The average execution time of each component in the background substitution system by the proposed fast method and after parallelization for ten 320x240 images on PAC Duo platform is shown in Table 2. From the first column, we can see that the system bottlenect lies in the component of solving the linear system of RW. After parallelization with the additional two DSPs, the time of "Linear System"

TABLE. 2. The average execution time (s) of each component in the background substitution system by the proposed fast method and the parallelization in ten 320x240 images on PAC Duo.

Component	Fast Algorithm	Parallel	Speedup Ratio
Face Detection	0.149s	0.149s	1.0
Gray Image	0.421s	0.421s	1.0
Down Sampling	0.006s	0.006s	1.0
Weight Calculation	0.032s	0.029s	1.1
Prior Model	0.110s	0.111s	1.0
Initial Guess	0.003s	0.003s	1.0
Linear System	3.790s	1.665s	2.2
Up Sampling	1.4309s	0.409s	3.49
Image Matting	0.138s	0.137s	1.00

can be reduced by nearly 2.2 times. It is also interesting that an optional down/upsampling approach can also improve the entire system performance. Fig. 6 shows some samples results of background substitution.

6. CONCLUSION

In this paper, a fast algorithm is proposed to speedup the process of solving the large linear system in Random Walk algorithm. Two tables, *TYPE* and *INDEX*, are created to fast calculate the closed-form solution. Then, the proposed method is parallelized to utilize the benefits of the architectural features. The experimental results show significant speed-up after applying the proposed method.

REFERENCES

[1] H. Wechsler and M. Kidode, "A Random Walk Procedure for Texture Discrimination," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-1, no. 3, pp. 272–280, 1979.

[2] Y. Lan, C. Li, Y. Zhang and X. Zhao, "A Novel Image Segmentation Method Based on Random Walk", PACIIA 2009, Asia-Pacific Conference on Computational Intelligence and Industrial Applications, Vol. 1, pp. 207-210, 2009.

[3] T. H. Kim, K. M. Lee, and S. U. Lee, "Edge-preserving Colorization Using Data-driven Random Walks with Restart," In Int'l Conf. Image Processing (ICIP), pp. 1641, 2009.

[4] R. Shen, I. Cheng, X. Li, and A. Basu, "Stereo Matching Using Random Walks," Proc. Int'l Conf. Pattern Recognition, 2008.

[5] T. H. Kim, K. M. Lee, and S. U. Lee, "Background Subtraction using Random Walks with Restart," The 12th International Workshop on Advanced Image Technology, 2009.

[6] T.-J. Lin, C.-N. Liu, S.-Y. Tseng, Y.-H. Chu and A.-Y. Wu, "Overview of ITRI PAC Project," in IEEE Int'l Symp. on VLSI Design, Automation and Test, 2008, pp. 188 –191.

[7] L. Grady. "Random walks for image segmentation," IEEE Transactions on Pattern Analysis and Machine Intelligence, 28(11):1768–1783, 2006.

[8] R. Merris, "Laplacian Matrices Of Graphs: A Survey," Linear Algebra and its Applications, 197, 198:143–176, 1994.

[9] R. Barrett and M. Berry, "Templates for the Solution of Linear Systems," 2nd ed. Philadelphia, PA: SIAM, 1994.